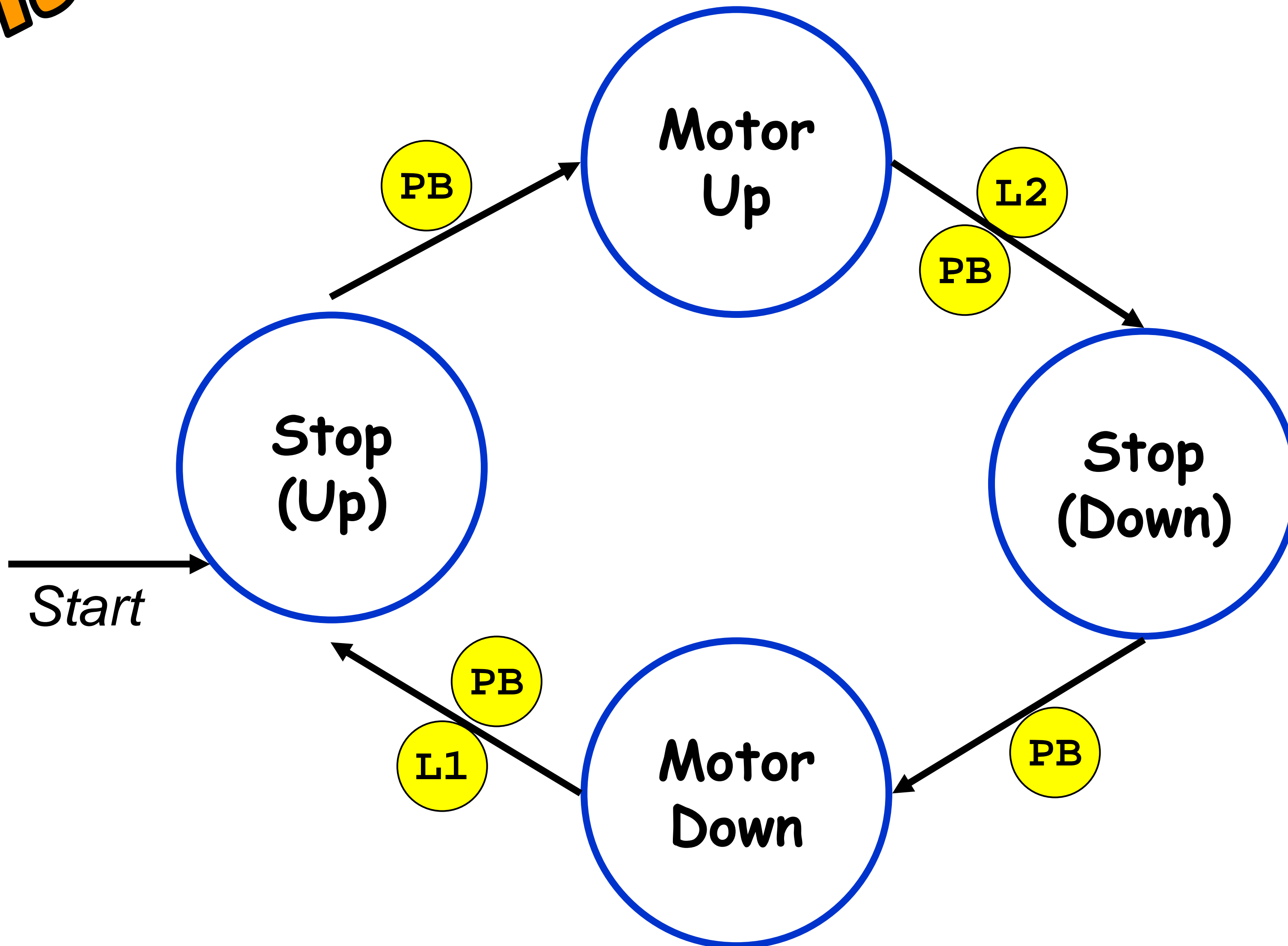
CS 3240

Presentation 4

Finite Automata

Recall

State Machine

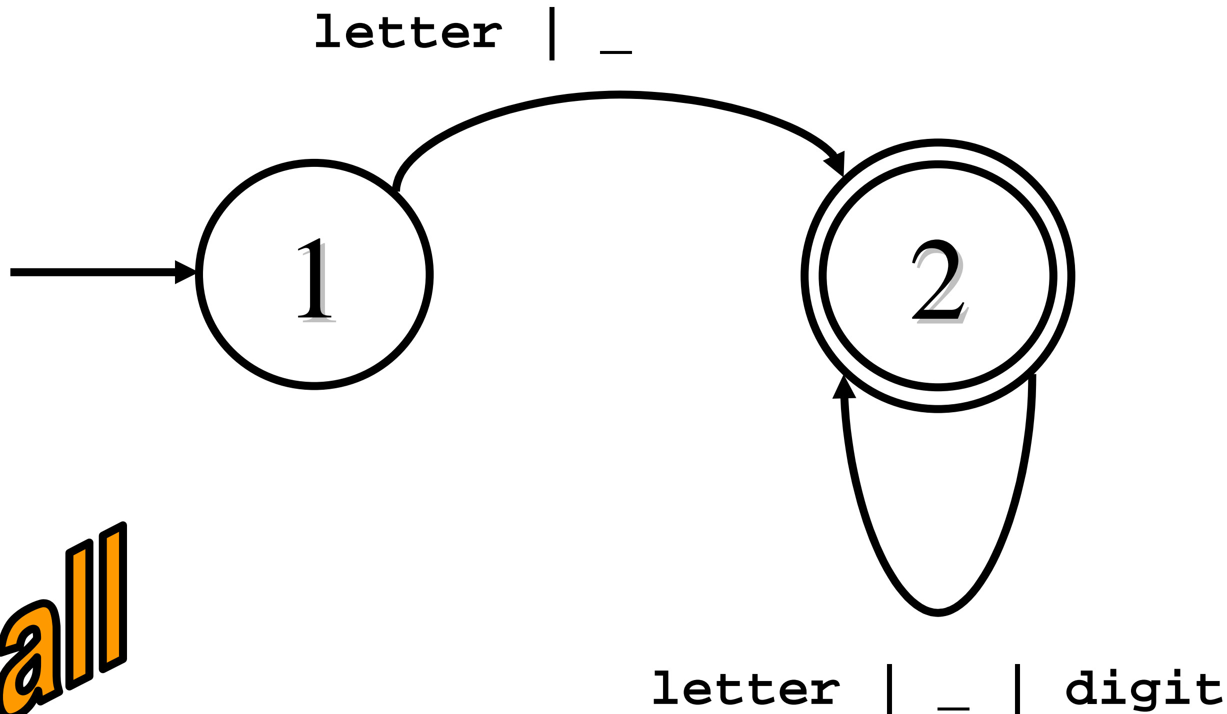


```
enum {L1, L2, PB, NONE} input;
enum {STOPUP, MOTORUP, STOPDOWN, MOTORDOWN} state;
while(1) {
   /* Get input here */
   switch(state) {
      case STOPUP:
         if(input == PB)
            state = MOTORUP;
         break;
      case MOTORUP:
         if (input == PB || input == L2)
            state = STOPDOWN;
         break;
      case STOPDOWN:
         if(input == PB)
            state = MOTORDOWN;
         break;
      case MOTORDOWN:
         if(input == PB || input == L1)
            state = STOPUP;
         break;
   }
   setMotor(state);
}
```

Warning!
Code not bulletproof
Do not use to control actual apparatus!

Previews of Coming Attractions

[a-zA-Z_][a-zA-Z_0-9]*



- Deterministic Finite Automata
- Regular Expressions
- Coding a scanner

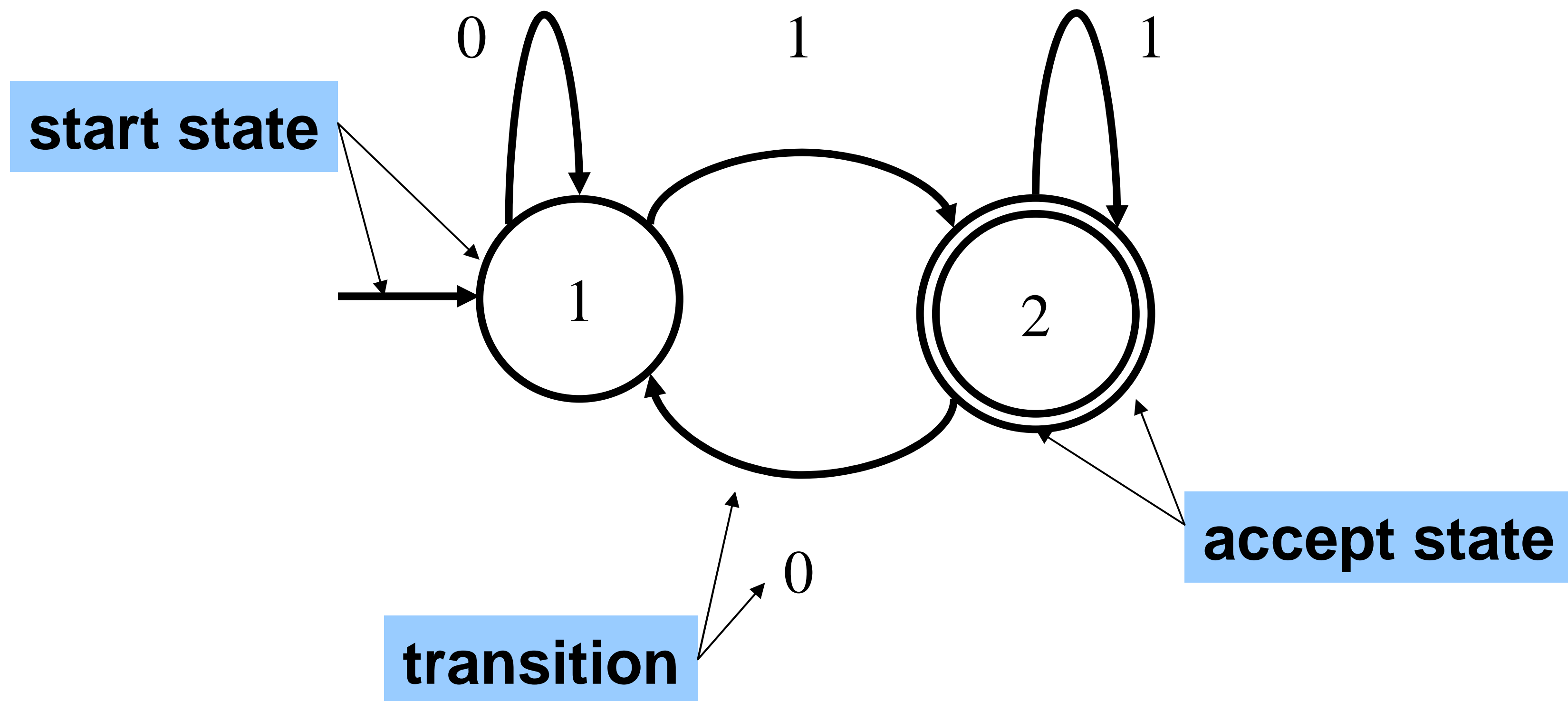
Deterministic Finite Automata

- Recall that theoreticians have developed a number of theoretical models to describe "computing"
  - Example: Turing Machine
- Simplest model is known as a DFA
- Deterministic: Machine will be in a state. Upon receipt of a certain symbol will go to a known state.
- Finite: The machines only have a certain number of states
  - The fascination here is that a machine with a finite number of states can "recognize" an infinite number of strings!
- Automata: (pl. of automaton) Cute little computing things

DFA's

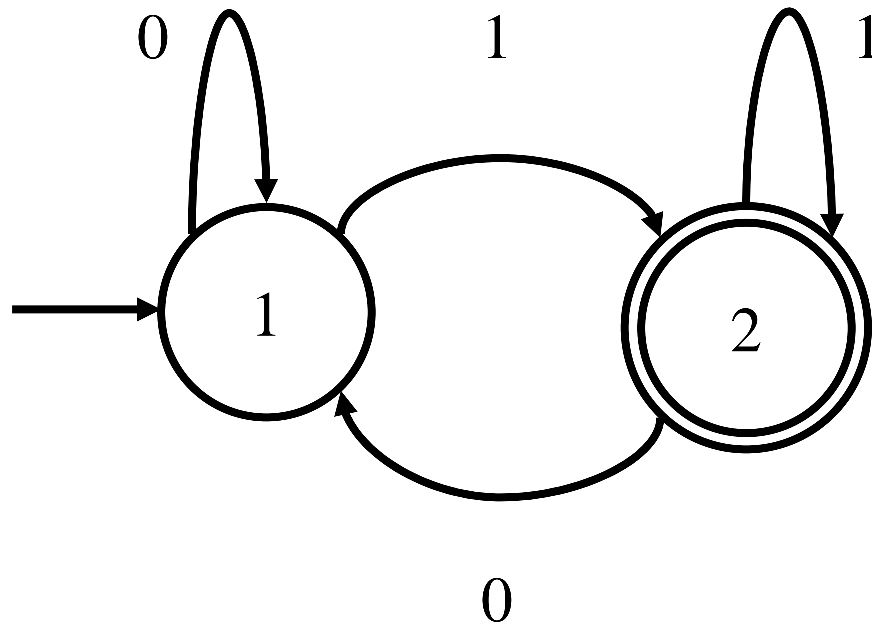
- DFA's recognize strings.
- If the input ends and the DFA is in an accept state then the string is "recognized"
- A "language" can be described as a set of strings
- A language is called a regular language if some finite automaton recognizes it.
- There is a precise mathematical definition of exactly what is meant by a finite automaton

Parts of a DFA



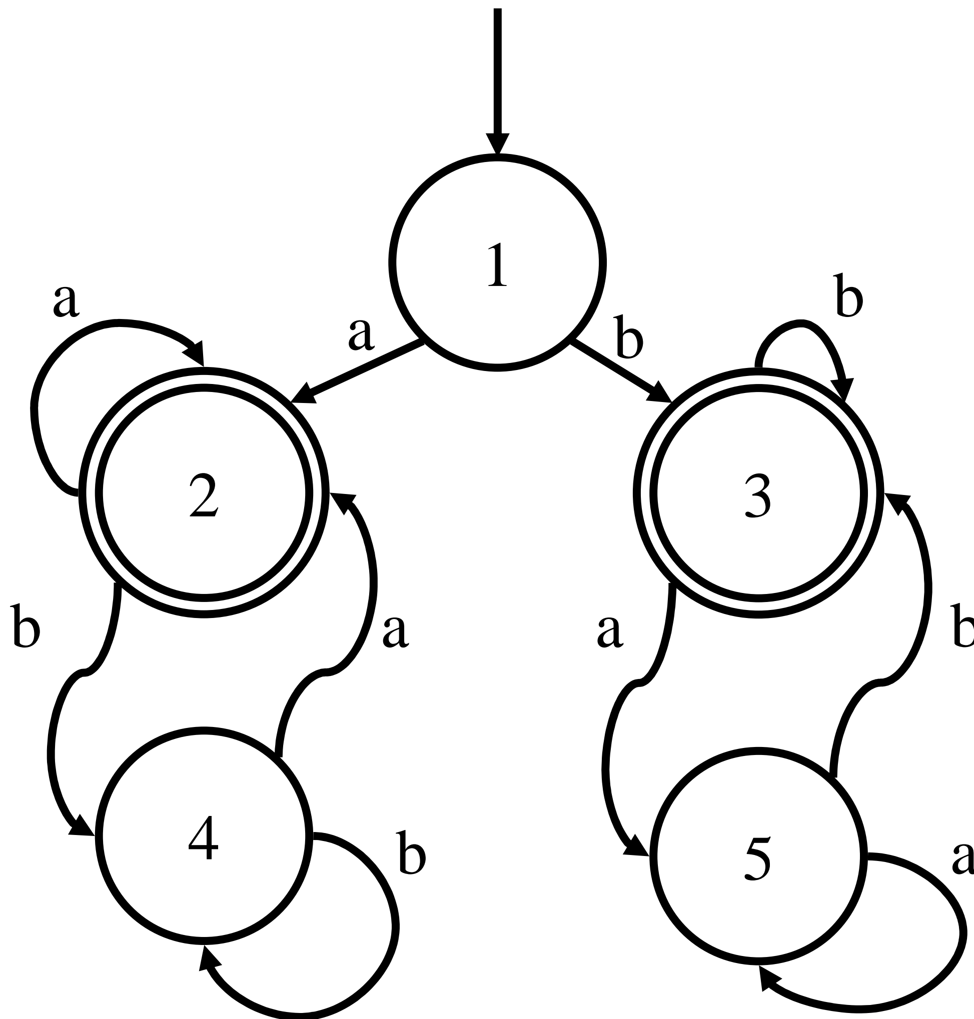
Note: The alphabet for this example is {0, 1}. Each state has a transition for every symbol in the alphabet

DFA Examples



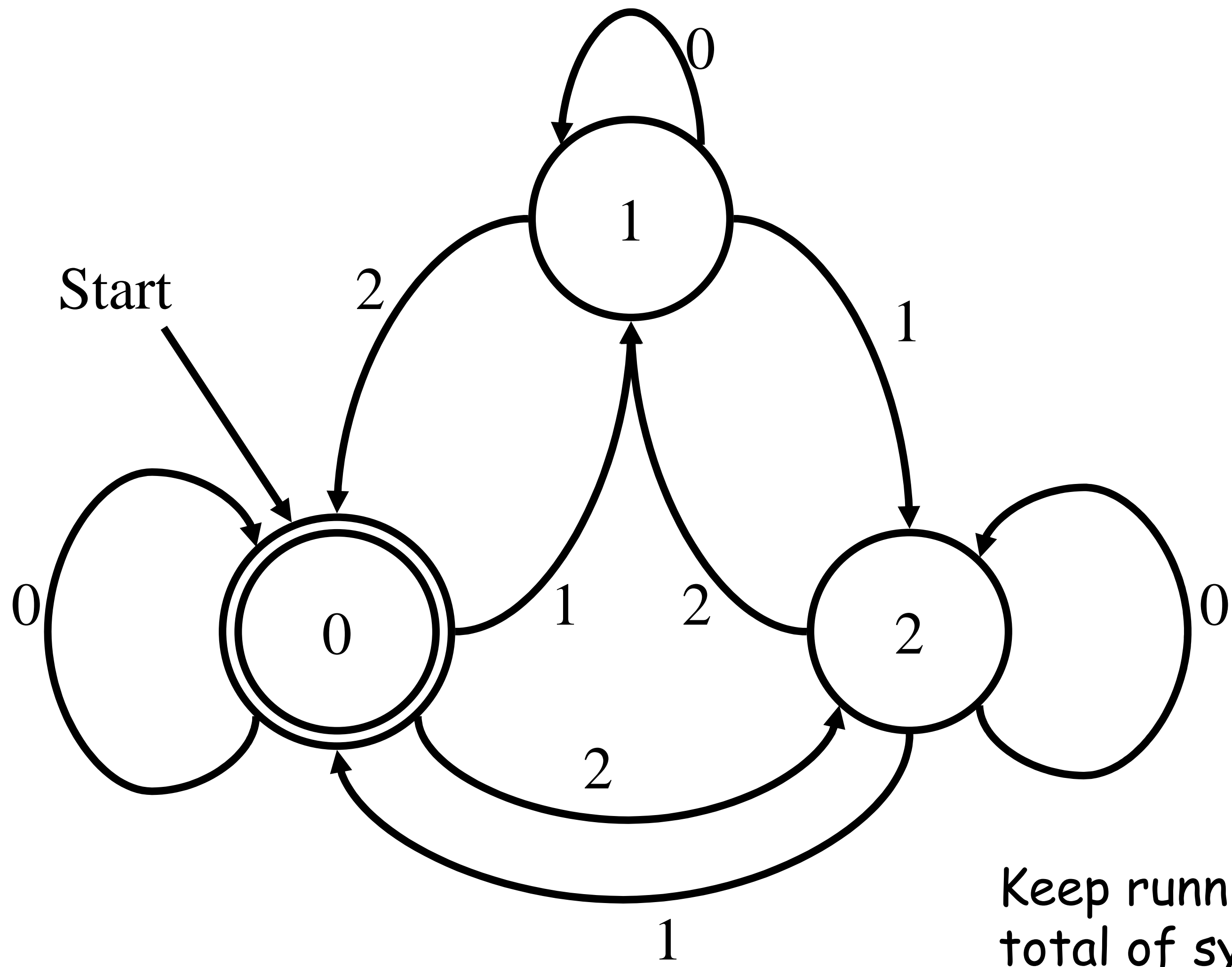
Accept all strings
that end in 1

DFA Examples



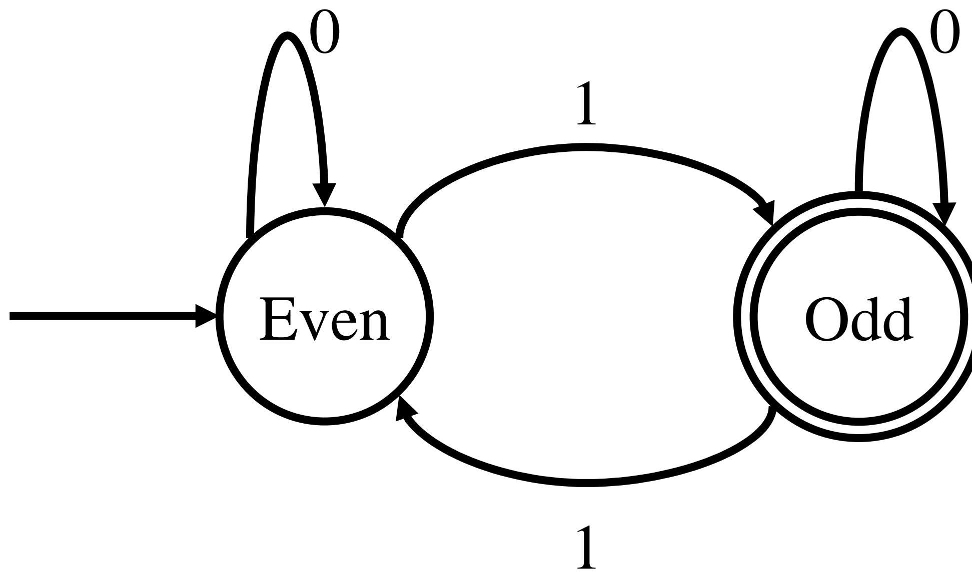
Accept strings of 'a's and 'b's that begin and end with same symbol

DFA Examples



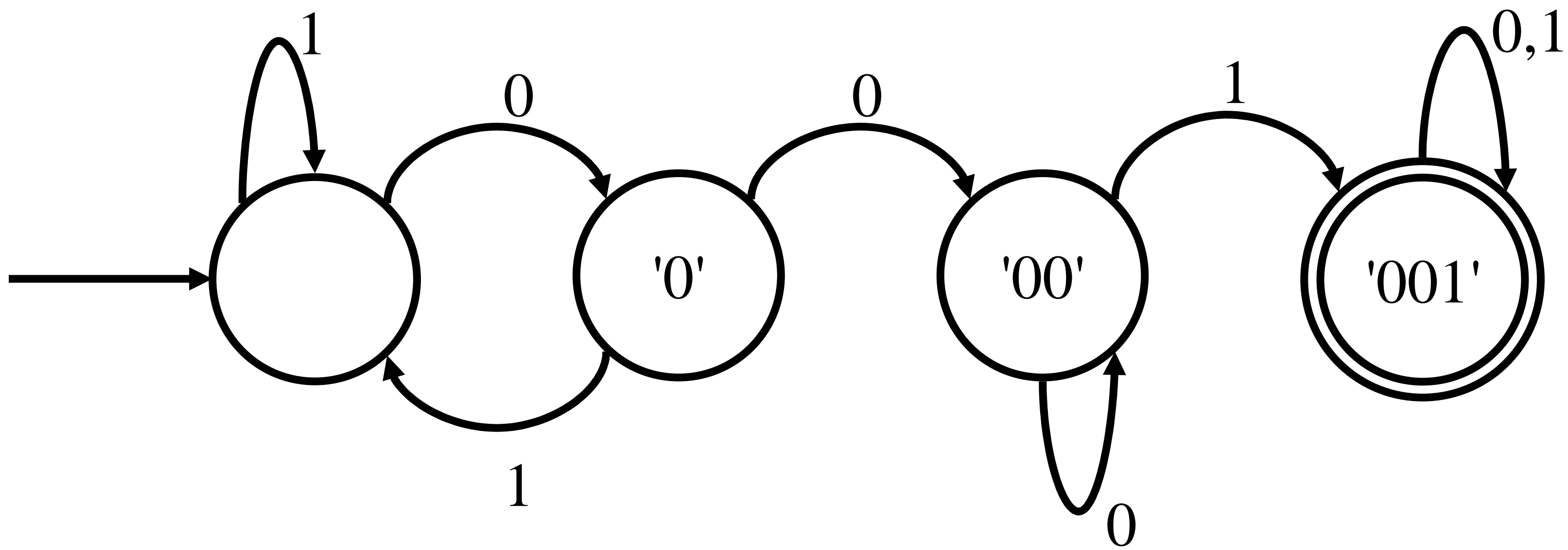
Keep running count of total of symbols read in mod 3. Accept on 0.

DFA Examples



Strings with an odd number of ones.

DFA Examples



Strings containing
the substring 001

Can DFA's be designed to accept any string?

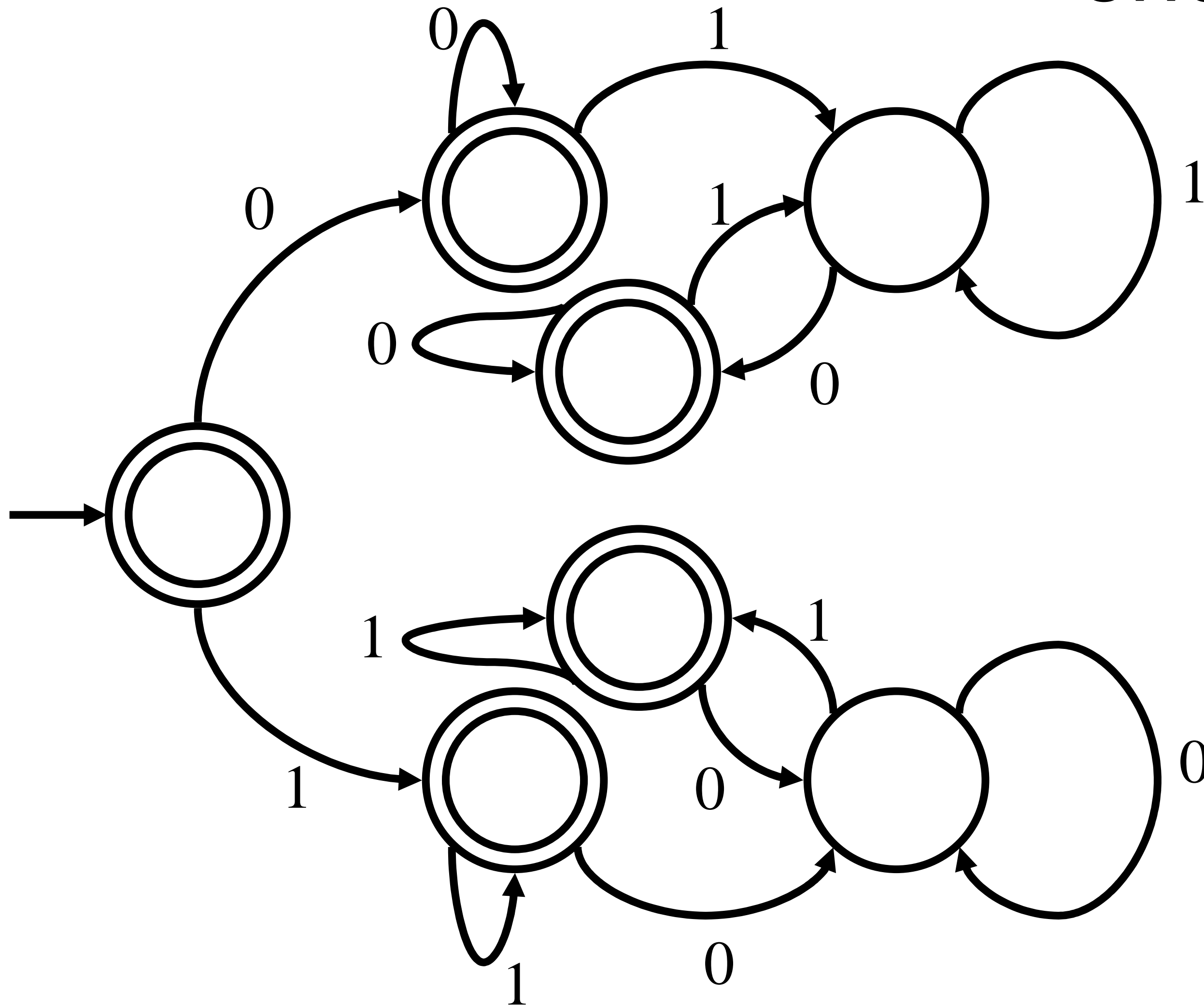
1-Yes

2-No

Examples

- Design a DFA to recognize strings that start out with k zeros followed by k ones.
- Design a DFA to recognize strings with an equal number of ones and zeros.
- Design a DFA to recognize strings with an equal number of strings "01" and "10". Impossible?
  - 1 yes
  - 2 No

Actually the third one is regular!



DFA to recognize strings with an equal number of strings "01" and "10"

DFAs, Regex

- DFA's are a mathematical concept representing a machine that can recognize strings in a language

- Languages recognizable by a DFA are regular

DFAs, Regex

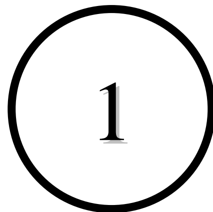
- Regular expressions also may be used to provide a description of a language
  - The value of the arithmetic expression (5+3)*4 is 32
  - The value of a regular expression is a language
- Regular expressions and DFA are equivalent
- Regular expressions are common in many CS apps

Lexical Analysis

State Machines

- A lexical analyzer is a state machine
- A state machine is a virtual or real device which responds to inputs with certain outputs that depend on what internal state the machine is in. This state is also changed as a result of the inputs.
- State machines are very similar to finite automata

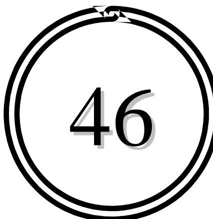
Symbology



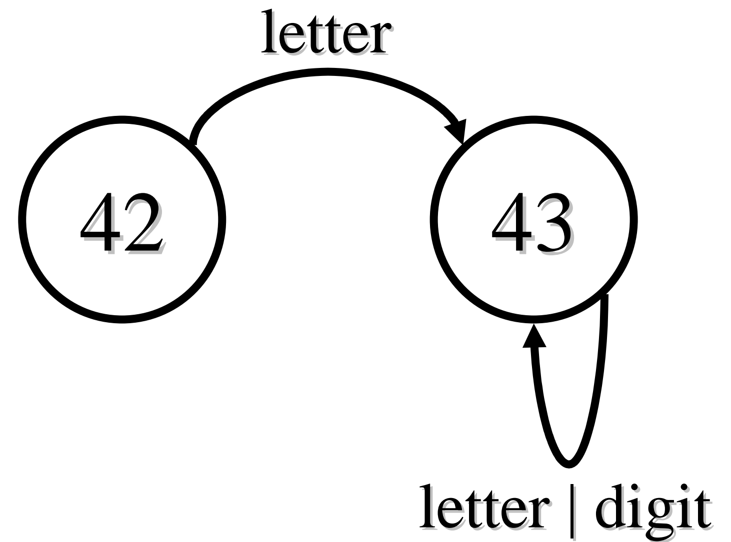
Start

Start
State

State
42



End
State

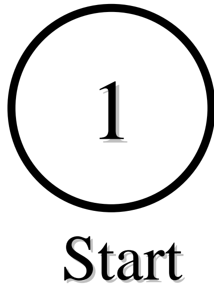


Typical State
Transitions

Problem

- Create a state machine that will recognize identifiers
- Error states omitted for simplicity

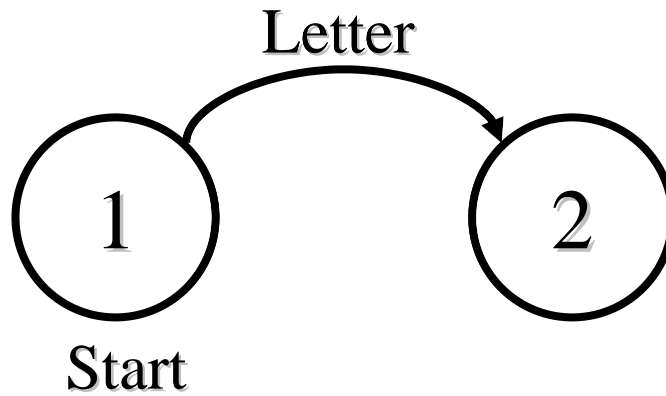
Recognize Identifiers*



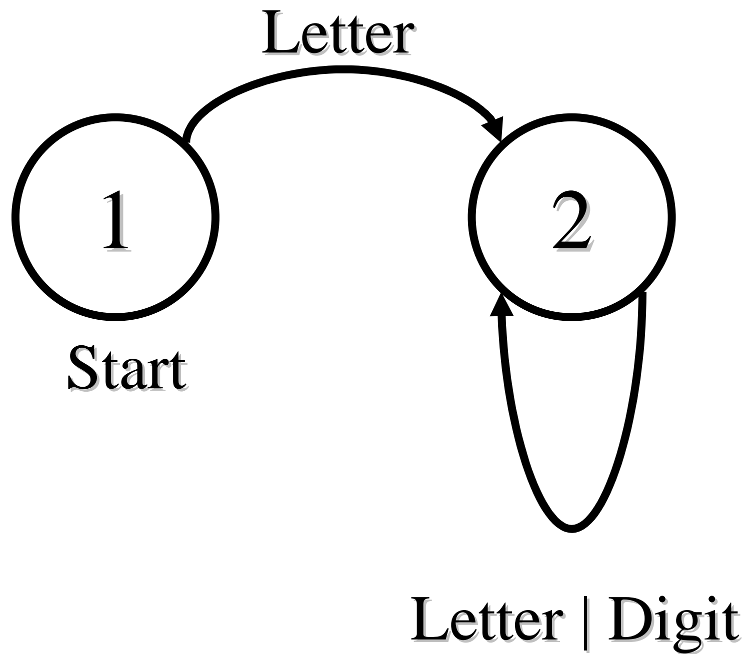
Begin is "Start State"
Read a character
Change state depending on character read

*Underscore omitted for simplicity

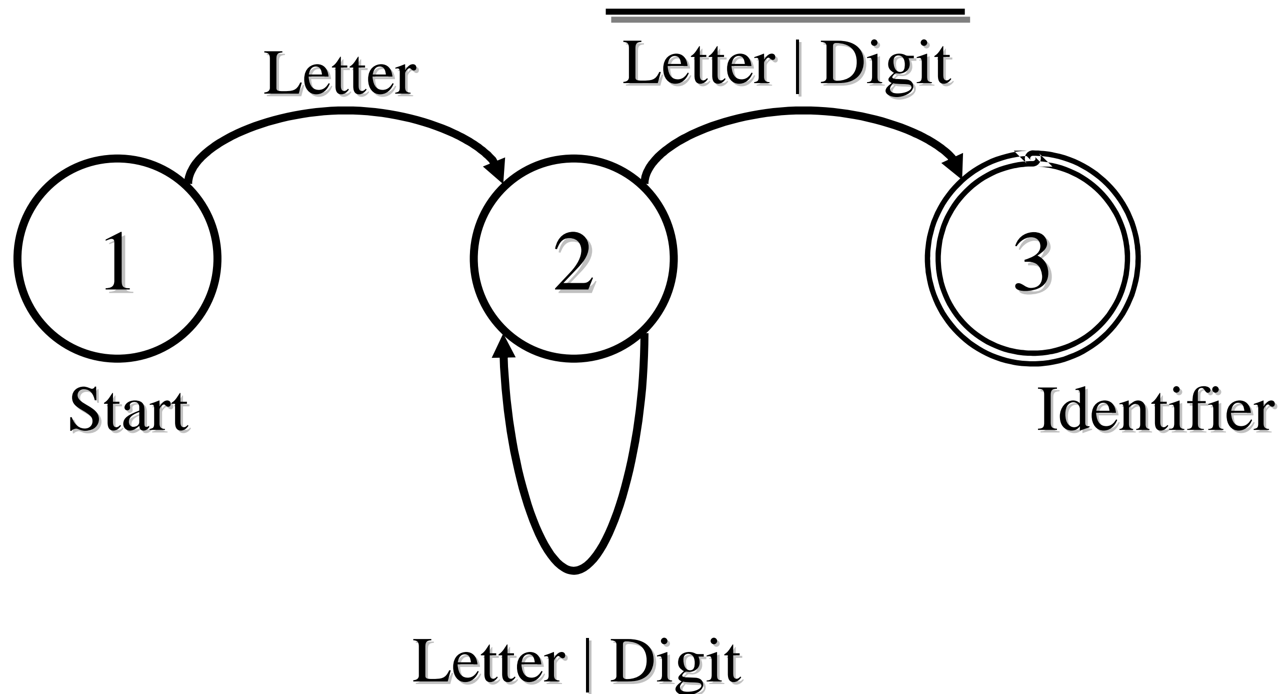
Recognize Identifiers



Recognize Identifiers



Recognize Identifiers



Notice the difference between this and pure finite automata

More Complex

- In a case like this:

```
   i       =        i       +       7          ;
<ident> <oper> <ident> <oper> <literal> <special>
```

- the white space characters terminate each identification but what about a case like this:

```
i=i+7;
```

- The character which terminates the identification process is part of the next token!

What to do?

- We need to save the last character (i.e. the one that put us in the end state)

- How?

- Push back on input
- Save a character in a variable

Whitespace

- Very little whitespace is actually required by c

```
main(){return(EXIT_SUCCESS);}
```

- would compile!

- There are some places where c does require whitespace

Lookahead

- In some languages the language specification is such that the determination of what token type is being scanned cannot be determined until well after the beginning of the token
- Example: FORTRAN
  - Variables don't have to be explicitly declared
  - Implicit typing based on first character
  - Real numbers begin with A-H or O-Z
  - Integers begin with I-N
  - (Mimicked typical mathematical convention)

Lookahead (continued)

- Fortran (continued)
- Typical loop construct

```
      DO 10 I = 1,10000
         ...
         ...
         ...
   10 CONTINUE
```

- meaning do all statements from the `DO` statement up to and including statement `10` starting `I` with a value of `1` and ending with a value of `10000` (incrementing by 1 as default)

Lookahead (continued)

- Fortran (continued)

- Spaces have no meaning so

  `DO 10 I = 1, 100000`

- would be equivalent to

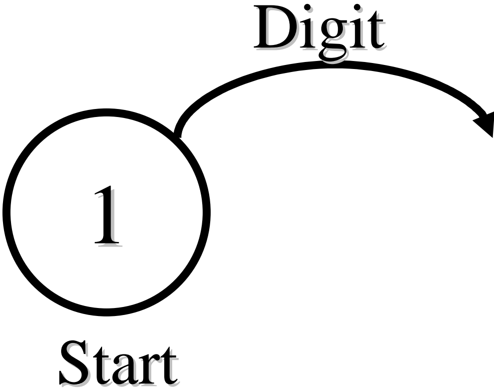
  `DO10I=1,100000`

- which would be differentiated from

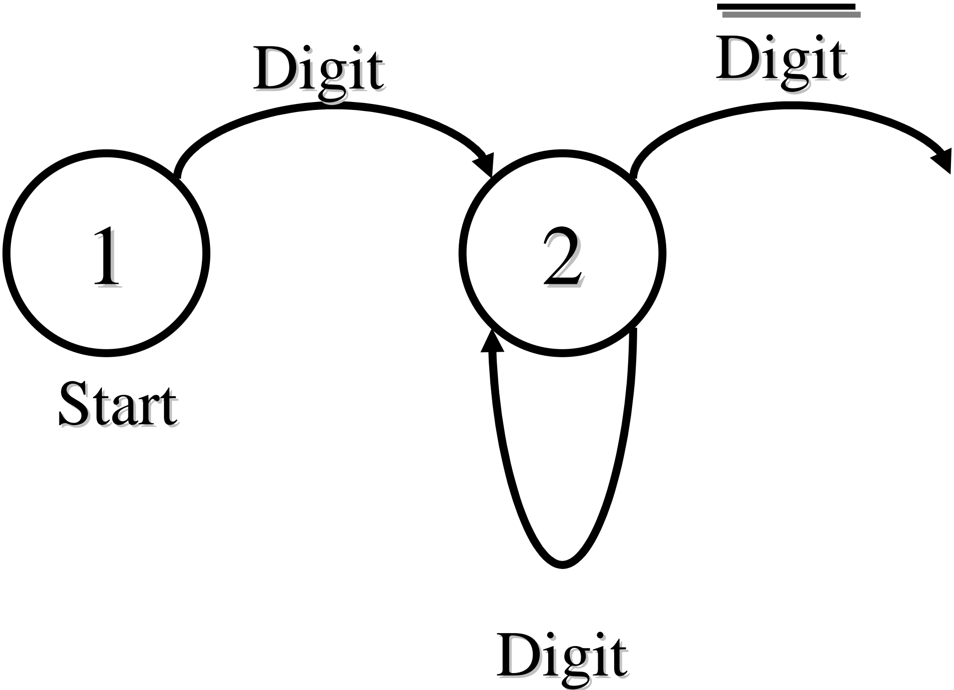
  `DO10I=1.100000`

- when the compiler encountered the **`,`** or **`.`**

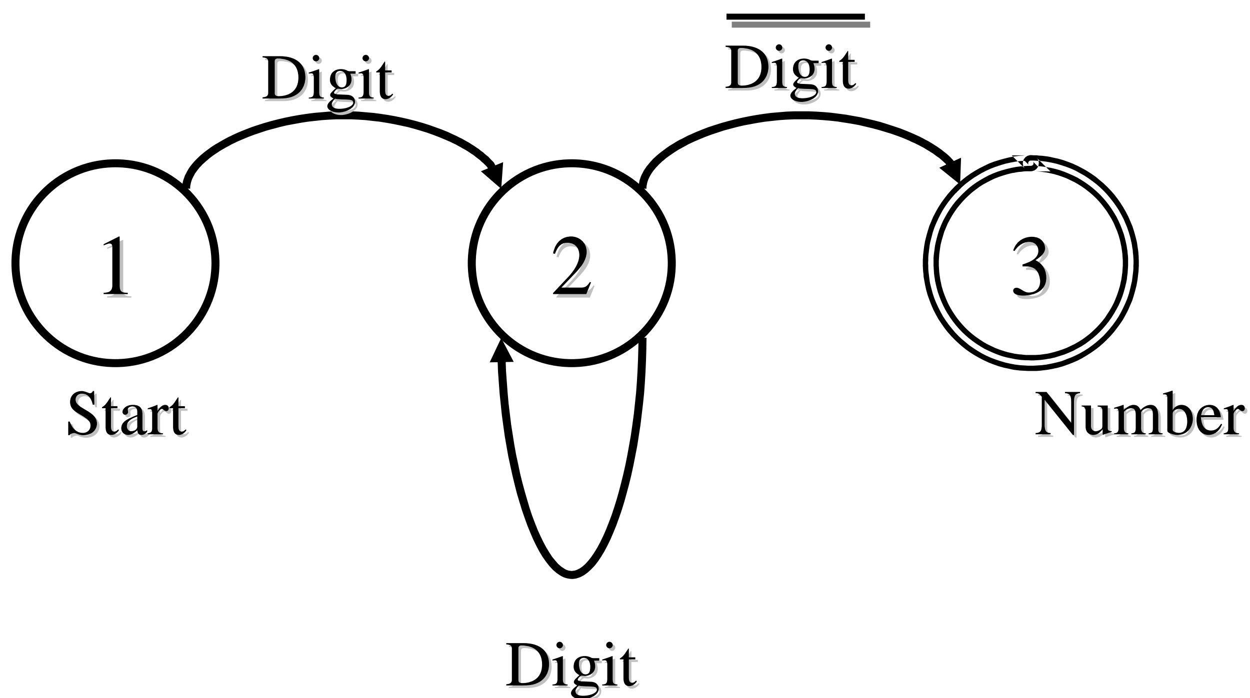
State Machine to Recognize Number



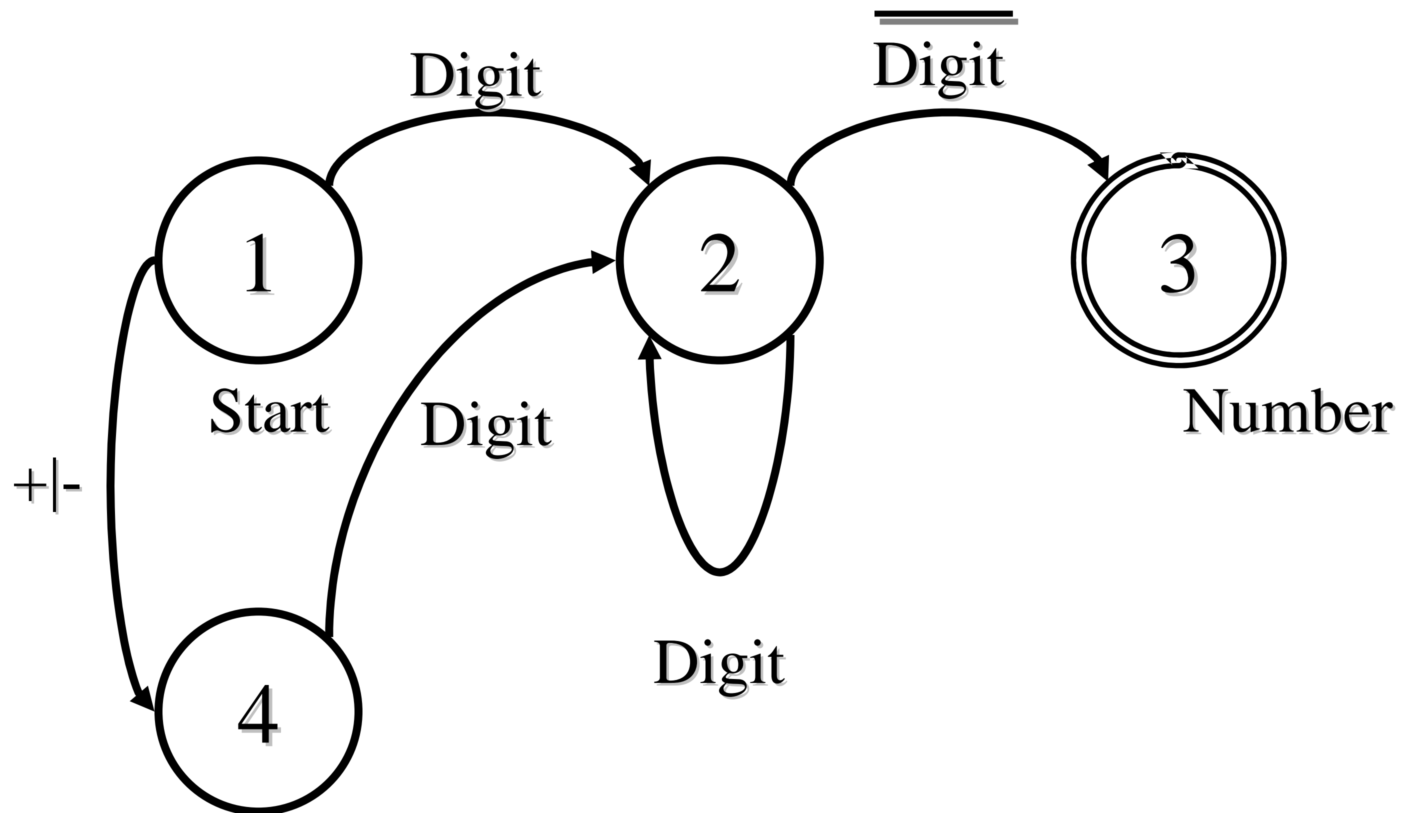
State Machine to Recognize Number



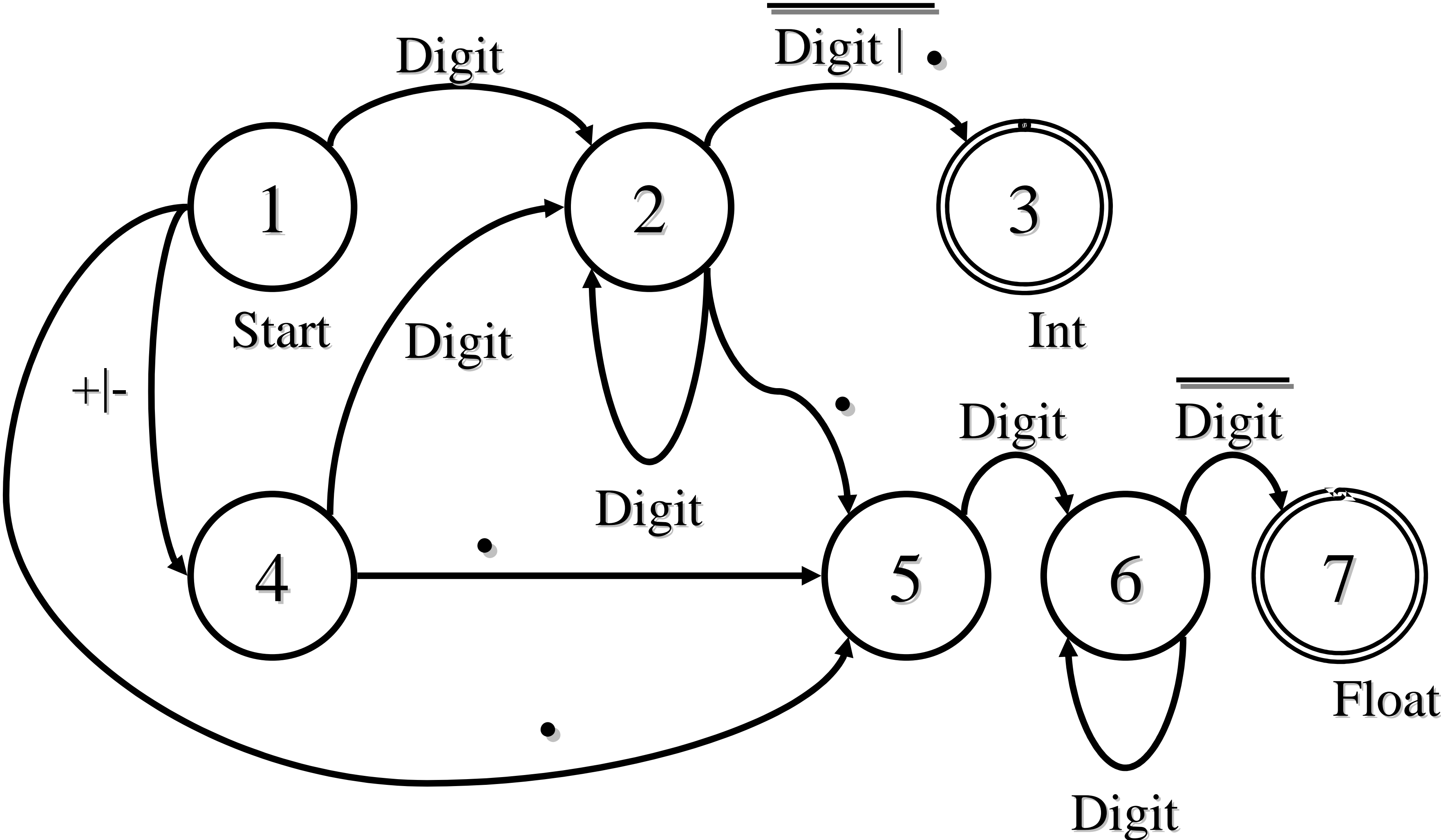
State Machine to Recognize Number



Modify to Recognize Sign



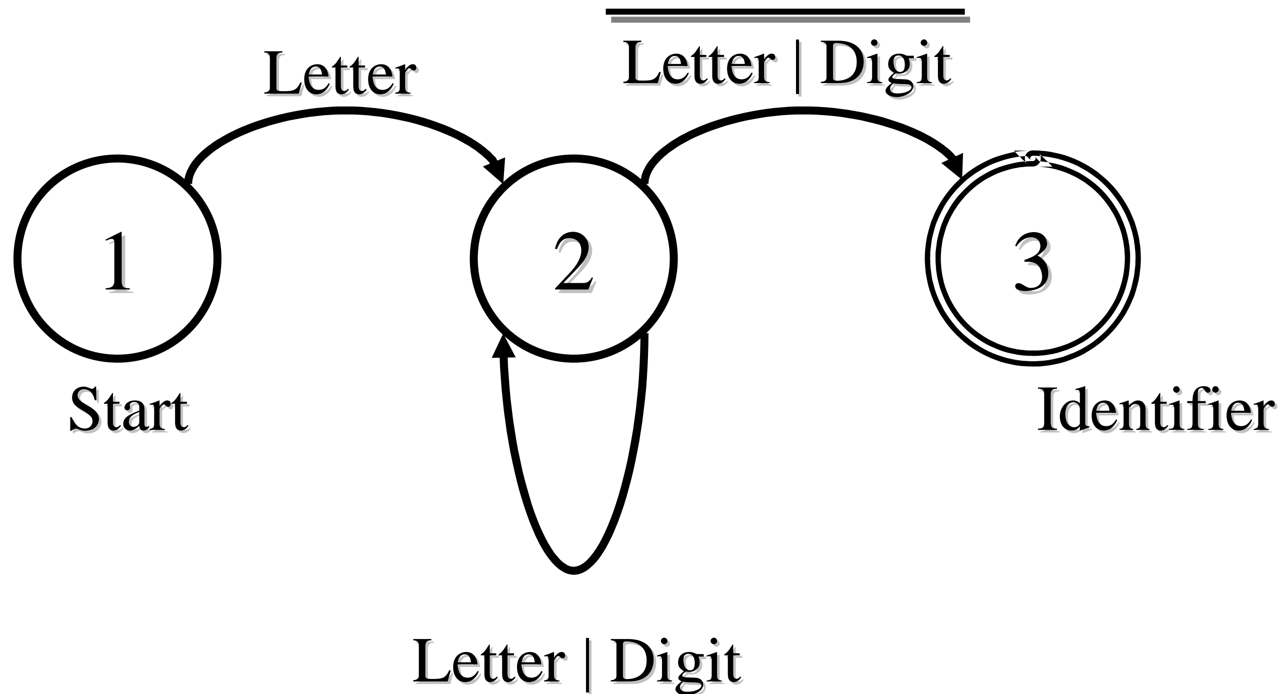
Modify to Recognize Decimals



Problem with State Machines?

- Drawings quickly get too big
- Need way to convert into code
- What is needed?
  - State variable
  - While Loop
  - Input a character
  - Switch

Recognize Identifiers



Example

```
enum {STATE1, STATE2, STATE3} state;
int inchar;
state = STATE1;
while((inchar=getchar())!=EOF) {
  switch(state) {
    case STATE1:
      if(inchar == 'a' || inchar == 'b' ||
         inchar == 'c' || inchar == 'd' ||
         inchar == 'e' || inchar == 'f' ||
         inchar == 'g' || inchar == 'h' ||
         inchar == 'i' || inchar == 'j' ||
         inchar == 'k' || inchar == 'l' ||
         /* How do you like it so far? */
```

Example

```
enum {STATE1, STATE2, STATE3} state;
int inchar;
state = STATE1;
while((inchar=getchar())!=EOF) {
  switch(state) {
    case STATE1:
      if((inchar >= 'a' && inchar <= 'z') ||
         (inchar >= 'A' && inchar <= 'Z' ))

      /* Better? */
```

Example

```
enum {STATE1, STATE2, STATE3} state;
int inchar;
state = STATE1;
while((inchar=getchar())!=EOF) {
  switch(state) {
    case STATE1:
      if(toupper(inchar) >= 'A' &&
         toupper(inchar) <= 'Z')

      /* Now we're cookin'??? */
```

Example

```
enum {STATE1, STATE2, STATE3} state;
int inchar;
state = STATE1;
while((inchar=getchar())!=EOF) {
  switch(state) {
    case STATE1:
      if(isalpha(inchar))
        state = STATE2;
      else
        /* ERROR!!! */
      endif
      break;
```

Example

```
case STATE2:
  if(isalpha(inchar) || isdigit(inchar))

  /* Here we go again! */
```

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else
    state = STATE3;
  break;
```

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else
    state = STATE3;
  break;
case STATE3:
  ungetc(stdin, inchar); /* Check return! */
  break;

/* Okay */
/* 1 - Yes */
/* 2 - No  */
```

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else
    state = STATE3;
  break;
case STATE3:
  ungetc(stdin, inchar);
  break;
```

Why not?

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else
    state = STATE3;
  break;
case STATE3:
  ungetc(stdin, inchar);
  break;
```

The character that put us in State 3 (the termination state) was encountered in State 2 processing

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else
    state = STATE3;
  break;
case STATE3:
  ungetc(stdin, inchar);
  break;
```

ungetc here

The character that put us in State 3 (the termination state) was encountered in State 2 processing

Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else {
    state = STATE3;
    ungetc(stdin, inchar);
  }
  break;

/* Depending on situation should be able to
   break out of loop upon reaching STATE3 */

/* Anything Missing??? */
```

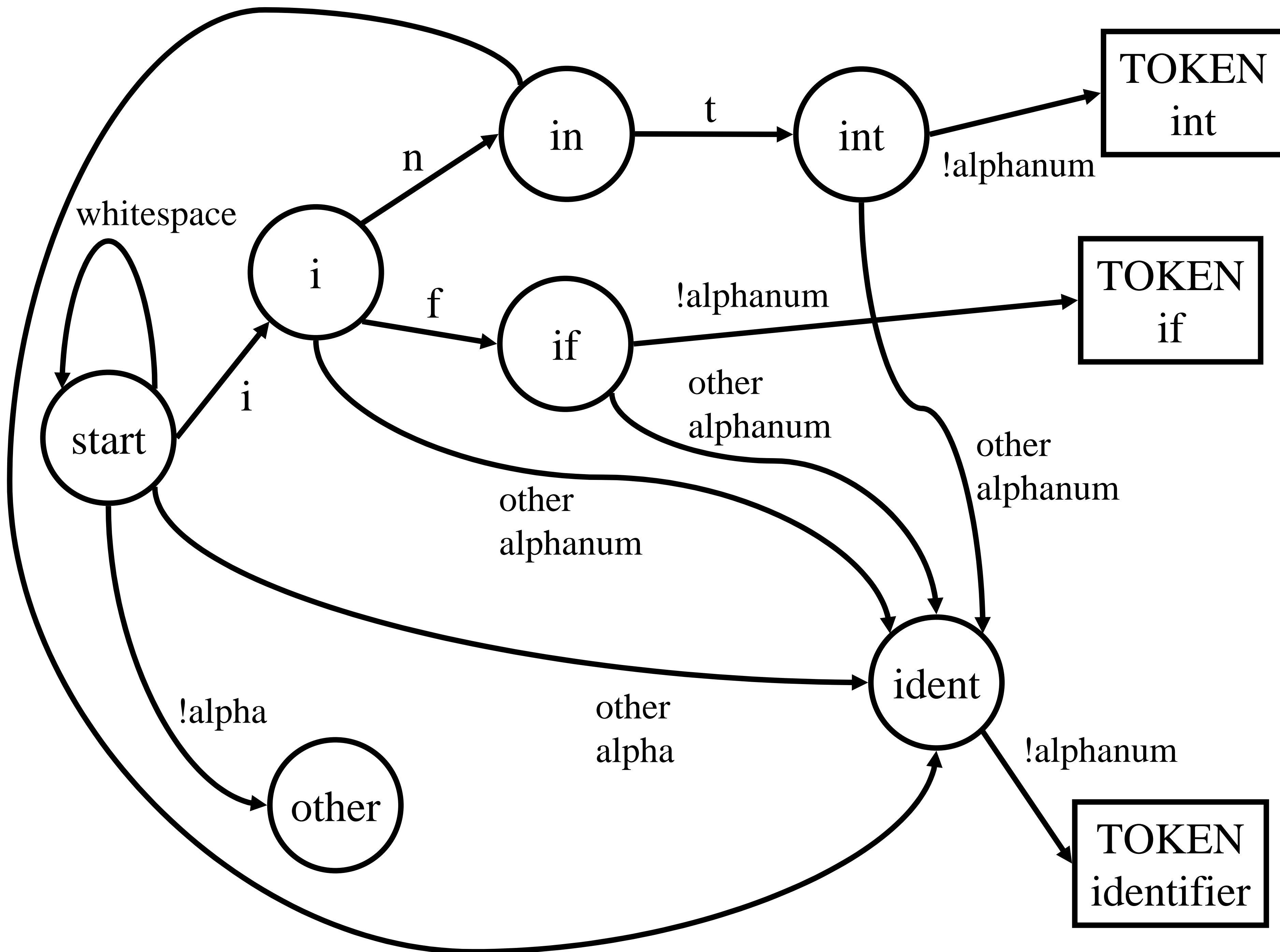
Example

```
case STATE2:
  if(isalnum(inchar))
    state = STATE2;
  else {
    state = STATE3;
    ungetc(stdin, inchar);
  }
  break;

default:
  /* Handle Error */
```

Longest Match Algorithm

```
While (input_char_read == (newline||white space||tab))
   /* Loop skipping white space */
Decide which token/tokens start with input_char_read
   Set flag(s) to note which tokens are potentially
   being recognized
While(input_char_read != char that matches no token
   being recognized)
   Adjust flags to reflect which token(s) are still
   being recognized
Push the character that matches no token back onto
   input stream
Return token to caller based on flag
```


TOKEN int
TOKEN if
TOKEN identifier
start
i
in
int
if
ident
other
whitespace
i
n
t
f
!alphanum
!alphanum
other alphanum
other alphanum
other alphanum
other alpha
!alpha
!alphanum

Lexical Analysis

- Tokens are assigned a number
- Each "class" of literal is assigned a number
  - int
  - float
  - string
  - etc
- A single code is assigned to all identifiers

Token Types

Category	Values					
• Operators	*	+	-	/	%	
	0	1	2	3	4	...
• Special	;	{	}	(	)	
	10	11	12	13	14	...
• Keywords	if	else	for	while		
	20	21	22	23		...
• Literals	42	3.141592	"Hello\n"			
	30	31	32			
• Identifiers	[_A-Za-z][_A-Za-z0-9]*					
	40					

Questions?

Questions?